VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Rozpoznání objektů v LiDARových datech

Object Recognition in LiDAR Data

2019 Filip Cima

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Filip Cima**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Rozpoznani objektu v LIDARovych datech Object Recognition in LIDAR Data
Jazyk vypracování:	čeština

Zásady pro vypracování:

Cílem práce je implementace detekce a rozpoznání objektů v LIDARových datech pomocí strojového učení.

Ve své práci proveďte:
1. Nastudujte způsob sběru a uložení dat z LIDARového skenru Velodyne.
2. Prostudujte dostupnou literaturu s tématikou detekce a rozponání objektů v LIDARových datech.
3. Ověřte dostupné modely, případně takový model navrhněte a realizujte ve frameworku pro strojové učení na datasetu.
4. Dosažené výsledky řádně zhodnoťte ve své práci.

Seznam doporučené odborné literatury:

[1] SZARVAS, Mate; SAKAI, Utsushi; OGATA, Jun. Real-time pedestrian detection using LIDAR and convolutional neural networks. In: Intelligent Vehicles Symposium, 2006 IEEE. IEEE, 2006. p. 213-218.
[2] PREMEBIDA, Cristiano, et al. A lidar and vision-based approach for pedestrian and vehicle detection and tracking. rn, 2007, 10: 2.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2018
Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 30. dubna 2019 .....

Rád bych na tomto místě poděkoval vedoucímu této bakalářské práce Ing. Janu Gaurovi, PhD. za poskytnutou pomoc a rady při tvorbě této práce.

Abstrakt

Práce vysvětluje základní principy umělých, konvolučních neuronových sítí. Popisuje problematiku **detekce objektů** z **LiDAR**ových dat pomocí **strojového učení**. Další část se věnuje popisu a srovnání dostupných prací zabývající se daným problémem. Poslední (praktická) část této práce je věnována popisu implementace metody **segmentace** a klasifikace s pomocí konvoluční neuronové sítě a knihoven TensorFlow a NumPy.

Klíčová slova: LiDAR, strojové účení, detekce, objekt, segmentace

Abstract

The thesis explains the basic principles of artificial convolution neural networks. It describes problems of **object detection** from **LiDAR** data using **machine learning**. The next part deals with the description and comparison of available works solving given problem. The last (practical) part of this work is devoted to the description of implementation of the **segmentation** and classification method using the convolutional neural network using TensorFlow and NumPy libraries.

Key Words: LiDAR, machine learning, detection, object, segmentation

Obsah

Seznam použitých zkratek a symbolů

ANN	–	Artificial Neural Network - Umělá neuronová síť
CNN	–	Convolutional Neural Network - Konvoluční neuronová síť
CRF	–	Conditional Random Field
CUDA	–	Compute Unified Device Architecture
DNN	–	Deep Neural Network - Hluboká neuronová síť
FLN	–	Feature Learning Network
LiDAR	–	Light Detection And Ranging
NP	–	NumPy - Knihovna jazyka Python pro práci s vektory, maticemi a vícerozměrnými poli
PC	–	Point Cloud
RADAR	–	Radio Detection and Ranging
ReLU	–	Rectified Linear Unit
RPN	–	Region Proposal Network
TF	–	TensorFlow - Knihovna jazyka Python pro strojové učení

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době máme mnoho nástrojů pro detekci terénu a mapování okolí. Jedním z nich je LiDAR - zařízení, které pracuje na základě vypočtení doby šíření laserového paprsku odraženého od okolních objektů. Je několik důvodů, proč je pro tuto problematiku LiDAR vhodný - např. vysoká hustota dat, spolehlivé fungování za tmy a rychlé zpracování vyslaných paprsků. V současné době autonomně řízených aut je použití LiDARového skeneru velice žádoucí. V kombinaci se stereo kamerami a radarem dokáže zaručit vysokou míru přesnosti rozpoznání objektů, které se vyskytují v bezprostřední blízkosti. Mezi další oblasti, kam LiDARové technologie směřují, patří také robotika a mapování těžko dostupných míst pomocí dronů. K hlavním průkopníkům patří přední automobilové koncerny a firmy vyvíjející autonomně řízená auta - jde například o společnosti *Toyota*, *Uber* nebo také *Google*.

Obrázek 1: Autonomně řízený automobil firmy Google s LiDARem fixovaným na střeše [1]

Tato práce se zabývá problematikou rozpoznání objektů za pomocí strojového učení z dat, která jsou získávány pouze z LiDARových zařízení. Základním stavebním kamenem praktické části této práce je knihovna *TensorFlow*, která poskytuje rozhraní pro využití algoritmů strojového učení. Je úzce spjata s další použitou knihovnou *NumPy*, která ulehčuje práci s vektory, maticemi a tensory.

V první části práce budeme podrobně rozebírat technologii LiDAR, uložení a zpracování dat z skenerů. Cílem je, aby se čtenáři seznámili se základními principy LiDARu, metodami, jak uložit a zpracovat informace, které zařízení produkuje. Jelikož máme k dispozici snímky poskytnuté zařízením firmy Velodyne, budeme se zabývat výhradně metodami přípravy dat pro

hardware této společnosti, nicméně metody se nebudou markatně lišit ani pro zařízení jiných firem.

Ve druhé části rozebereme principy neuronových sítí, popíšeme základní typy neuronových sítí a rozebereme jejich učení. Zejména pak pomocí *algoritmu zpětného šíření chyby*. Rozebereme také často používané techniky v hlubokých neuronových sítích, které přispívají k lepším výsledkům při učení.

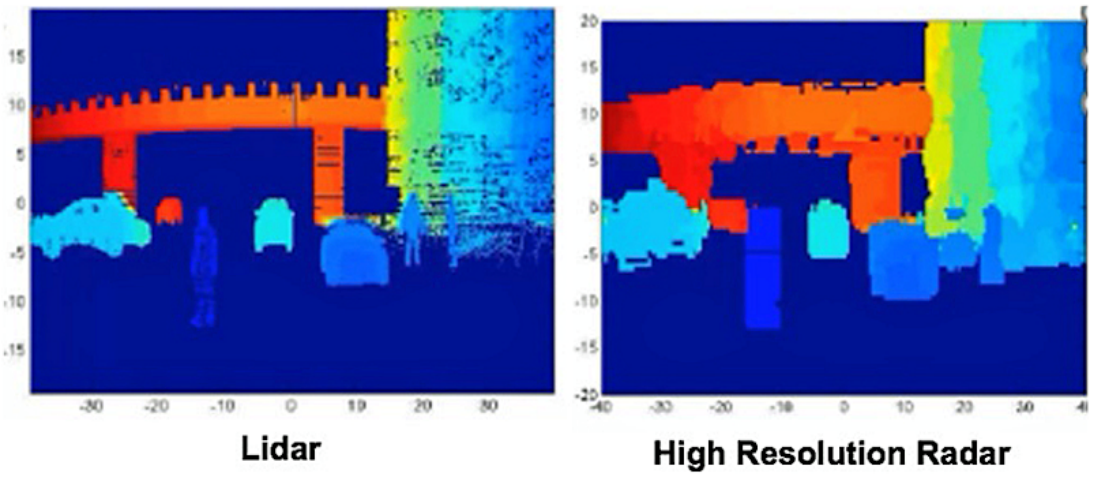
Třetí část se bude věnovat problematice konvolučních neuronových sítí. Popíšeme jejich silné stránky. Vysvětlíme, k řešení jakých problémů jsou užitečné, popíšeme základní, často používané, vrstvy sítě a ukážeme, jak by přibližně měla vypadat struktura sítě.

Čtvrtá část se bude zabývat shrnutím dostupných možností rozpoznávání objektů z LiDA-Rových dat pomocí konvolučních neuronových sítí a jiných metod užitých v předzpracování dat. Primárně bude zaměřena na srovnání dostupných prací založených na takovýchto sítích, ale zřetel bude brán i na jiné metody - např. shlukování.

V páté a zároveň poslední části se budeme věnovat analýzou a finální implementací segmentace objektů z dat KITTI [2] včetně předzpracování dat do podoby vhodné pro konvoluční neuronovou síť. Rozebereme také přesnost a rychlost detekce objektů. Představíme jednotlivé vrstvy použitého modelu sítě a zmíníme experimenty, které by teoreticky mohly zvýšit přesnost detekce.

2 Technologie LiDAR a jeho data

LiDAR je metoda měření doby šíření světelného (laserového) paprsku odraženého od objektu. Oproti RADARu technologie disponuje mnohem rychlejším zpracováním paprsku, jelikož je šíření světelného paprsku mnohonásobně rychlejší než rádiových vln. Dále LiDAR operuje s kratšími vlnovými délkami a tím je schopen vytvořit mnohem přesnější 3D mapu objektu než RADAR.



Obrázek 2: Porovnání přesnosti LiDARu a RADARu [3]

2.1 Specifikace zařízení Velodyne HDL-64E

Přístroj Velodyne HDL-64E je vlajkovou lodí mezi LiDARovými skenery v současnosti. Je používán v autonomních řídících systémech, protože poskytuje výborné rozlišení výsledné mapy terénu na základě vysokého počtu laserů v samotném zařízení. Poměrně nedávno představila firma Velodyne zařízení *VLP-16*, které je cenově dostupné, avšak nemá tak kvalitní výstup. To může mít za důsledek zhoršení přesnosti při klasifikaci a segmentaci objektů za pomocí strojového učení, či jiných metod počítačového vidění [4].

2.1.1 Senzor

Konkrétní zařízení disponuje celkem 64 laserovými emitory a přijímači, které jsou umístěny v rotující hlavě. Zorné pole tohoto modelu je 360° pro azimut a 26,8° pro elevaci. Přibližný svislý úrovňový rozestup mezi paprsky je 0,4°. Frekvence otáčení hlavy kolem své osy se pohybuje v intervalu 5 až 15 Hz. Funkční vzdálenost paprsků je až 50 metrů pro chodníky (cca 0,10 intenzita reflektovaného paprsku) a až 120 metrů pro auta a lesklé povrchy (cca 0,80 intenzita reflektovaného paprsku). Standardně operuje zařízení v teplotách od -10°C až do 50°C [4].

Použité lasery jsou zařazeny do kategorie 1 podle standardu IEC 60825-1, což znamená, že jsou pro lidské oko naprosto bezpečné. Paprsky jsou rozděleny do čtyř bloků a operují s vlno-

vou délkou 905 nm. Nicméně jsou evidovány případy, kdy paprsek nenávratně poškodil čočku fotoaparátu [5].

Obrázek 3: Velodyne LiDAR HDL-64E [4]

2.1.2 Mechanické vlastnosti

Tento LiDAR vyráběný společností Velodyne neváží více než 13 kilogramů, je napájen pod napětím $15V \pm 1,5V$ a disponuje ochranou IP 67 dle normy IEC 529. To znamená, že je prachuvzdorný a voděodolný do hloubky jednoho metru až na 30 minut, což je naprosto dostačující ochrana pro naše případy užití LiDARu [4].

2.1.3 Výstup

Pro transport dat ze zařízení jsou používány 100 MB/s ethernetové UDP pakety, které jsou odchytávány po síti a dále zpracovávány. Odchycená data mohou být zpracovány v reálném čase, nebo mohou být uloženy do souborů (například s příponou *.pcap*) a tím si otevíráme možnosti pečlivější a časově náročnější analýzy dat [4].

2.2 Zpracování výsledných paketů

Každý paket se skládá celkově z 1248 bajtů. První blok 42 bajtů je vyhrazen hlavičce paketu. Dále následuje 12 stobajtových bloků, kde každému náleží identifikační číslo bloku, rotační informace úhlu azimutu θ. Každý blok má úhel inkrementovaný o 2° oproti předchozímu. Zbylých 96 bajtů každého bloku je rozděleno mezi 32 podbloků, které nesou informace o vyslaných paprscích, tedy o intenzitě a vzdálenosti. Posledních 6 bajtů je vyhrazeno údajům o čase a verzi firmware zařízení [4].

1248 bajtů						
42 bajtů	**1200 bajtů**					**6 bajtů**
Hlavička paketu	**100 bajtů**	**100 bajtů**		**100 bajtů**	**100 bajtů**	Status
	Datový blok 1	Datový blok 2	...	Datový blok 11	Datový blok 12	Časové razítko - 4 B
	ID bloku - 2 B	ID bloku - 2 B		ID bloku - 2 B	ID bloku - 2 B	Verze firmware - 2 B
	Rotační informace - 2 B	Rotační informace - 2 B		Rotační informace - 2 B	Rotační informace - 2 B	
	96 bajtů	**96 bajtů**		**96 bajtů**	**96 bajtů**	
	32x laserové info	32x laserové info	...	32x laserové info	32x laserové info	
	Vzdálenost - 2 B	Vzdálenost - 2 B		Vzdálenost - 2 B	Vzdálenost - 2 B	
	Intenzita - 1 B	Intenzita - 1 B		Intenzita - 1 B	Intenzita - 1 B	
			...			

Obrázek 4: Struktura paketu přijatého ze zařízení Velodyne HDL-64E

S informacemi zjištěnými z paketů dostaneme snadno informace o sférických souřadnicích jednotlivých bodů. Každá sférická souřadnice je definována jako n-tice P

$$P = (r, \theta, \varphi), \tag{1}$$

kde r je vzdálenost bodu od LiDARového senzoru, θ je úhel azimutu a φ je úhel zenitu. Vzdálenost r vychází ze vztahu

$$r = \frac{tc}{2}, \tag{2}$$

kde t je čas od vyslání paprsku až do přijetí LiDARem a c je rychlost světla. Takto vypočtená hodnota musí být vydělena dvěma, jelikož paprsek vzdálenost od bodu podnikne dvakrát. A takto definované sférické souřadnice můžeme lehce převést do kartézské soustavy souřadnic pomocí vztahů

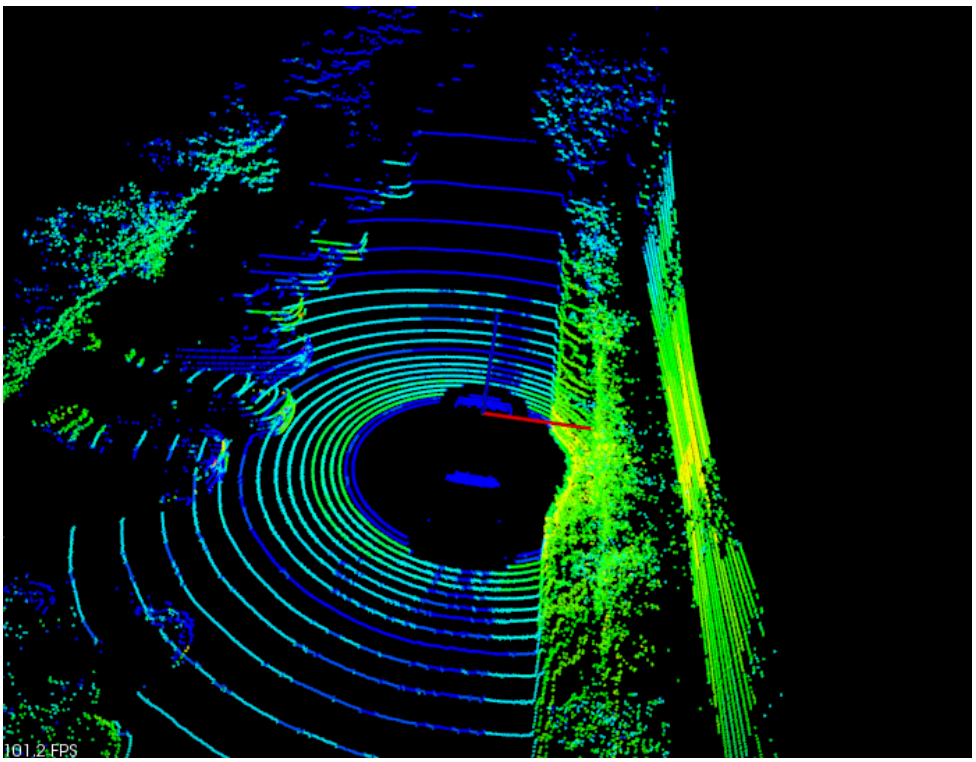
$$\begin{aligned} x &= r \sin\theta \cos\varphi \\ y &= r \sin\theta \sin\varphi \\ z &= r \cos\theta \end{aligned} \tag{3}$$

kde x, y, z jsou výsledné kartézské souřadnice bodu v trojrozměrném prostoru, r je vzdálenost bodu od LiDARu, θ je úhel azimutu a φ je úhel zenitu [6].

2.2.1 Zpracování pomocí knihovny PCL

Pro zpracování se dnes často používá knihovna PCL (Point Cloud Library) napsaná v jazyce C++, která poskytuje rozhraní k práci s LiDARovými zařízeními firmy Velodyne. Pro načtení dat je vhodné využít třídy HDLGrabber, následná vizualizace je realizována pomocí třídy *Sim-*

pleHDLViewer. Z důvodu úspory místa se informace získané z LiDARu ve většině případů zapisují do binárních souborů, mezi další častý používaný formát řadíme PCD (Point Cloud Data), který spadá pod výše zmiňovanou knihovnu PCL a jeho hlavní výhodou je čitelnost, jelikož data nejsou zapisována binárně, ale v textovém formátu. Našim účelům postačí zapisovat pouze informace o bodech z mračna, ale běžně se do souboru uvádí meta informace o zařízení, ze kterého body pochází.



Obrázek 5: Vizualizace LiDARových dat pomocí knihovny PCL [7]

Do binárního souboru zapisujeme pouze čtyřbajtová čísla, konkrétně se jedná o souřadnice X, Y, Z a intenzitu paprsku. Získané body zapisujeme sekvenčně za sebou, dokud neprojdeme všechny z mračna. Ve výsledku pak bude velikost binárního souboru, který obsahuje celkem 150 000 bodů, přibližně 2,29 MB. V opačném případě zápisu do textového souboru formátu *PCD*, by stejně obsáhlé mračno bodů zabralo 5,3 MB.

Formát	**Čitelný**	**Úsporný**	**Může obsahovat metadata**
.PCD	ANO	NE	ANO
.BIN	NE	ANO	ANO

Tabulka 1: Srovnání formátů uložení LiDARových dat

3 Neuronové sítě

V této kapitole se věnujeme teoretickému úvodu neuronových sítí. V první části se zabýváme podrobným popisem neuronu, který je základním stavebním prvkem takovýchto sítí. Dále se zabýváme architekturou konvolučních neuronových sítí a poslední část sekce je věnována algoritmu backpropagation.

3.1 Úvod do neuronových sítí

Umělé neuronové sítě spadají mezi výpočetní systémy inspirované chováním biologických neuronů mozků živočichů. Sice se nyní může spojení mezi biologickými a umělými neurony zdát nereálné, ale později v práci budou popsány principy fungování neuronových sítí, které připomínají funkci těch živočišných. Neuronová síť je definována jako uspořádaná šestice

$$S = (N, C, I, O, w, t), \tag{4}$$

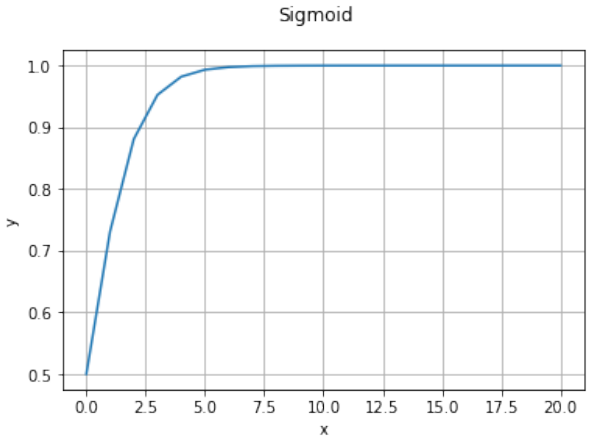
kde N je množina všech neuronů sítě, C je množina spojů mezi neurony, I je množina vstupních neuronů, O je naopak množina výstupních neuronů, w a t jsou funkce váhové a prahové [8].

3.1.1 Neuron

Neuron je nutným základem každé neuronové sítě. Skládá se z vektoru vstupů (x_1 až x_n) a vektorů vah (w_1 až w_n). Jednotlivé váhy spadají do oboru reálných čísel, pokud jsou záporné, projevují své inhibiční vlastnosti. Takzvaná vážená suma vstupů je definována podle následujícího vztahu

$$y_i n = bias + \sum_{i=0}^{n} x_i w_i, \tag{5}$$

kde x_1 až x_n jsou vstupy neuronu, w_1 až w_n jsou synaptické váhy vstupů a $bias$ je vstup, který zastává hlavní vstup neuronu. Výstup z neuronu získáme dosazením výsledné hodnoty do aktivační (přenosové) funkce. Mezi nejdůležitější a nejčastěji vyskytované aktivační funkce neuronové sítě patří takzvané sigmoidální funkce. Tyto funkce se používají, protože se dají lehce zderivovat, a to snižuje výpočetní čas učení sítě. U aktivačních funkcí je často vyžadován výstup v intervalu $\langle 0, 1 \rangle$, ale existují také bipolární funkce, jejíž výstup je v intervalu $\langle -1, 1 \rangle$ [9].



Obrázek 6: Průběh aktivační funkce sigmoid

3.1.2 Perceptron

Perceptron je nejjednodušším modelem neuronové sítě. Skládá se pouze z jednoho neuronu a je navržen pro řešení triviálních problémů. Přesněji řečeno, síť typu perceptron je použitelná pouze pro řešení problému s lineárně separovatelnými množinami. Nicméně sítě složené z více perceptronů dokáží řešit širší pole problémů [9].

3.1.3 Vstupní data

Stejně jako se lidé a jiné živé bytosti učí z příkladů, tak i k naučení umělé neuronové sítě metodou s učitelem je potřeba množina trénovacích dat. Za trénovací množinu považujme množinu prvků M definovaných dvojicemi vektorů vstupních a výstupních dat

$$M = \{\{I_1, O_1\}, \{I_2, O_2\}, ..., \{I_n, O_n\}\}, \tag{6}$$

kde M je množina dat pro třenování, I_i je vektor vstupů a O_i je vektor požadovaných výstupů neuronové sítě.

3.2 Učení sítě

Po získání a připravení dat je potřeba model sítě naučit. Obecně je cílem všech učících algoritmů nastavit váhy tak, aby se minimalizovala celková chyba a síť by pak vykazovala velice dobré výsledky. Učení můžeme rozdělit do dvou skupin, a to *učení s učitelem* (supervised learning) a *učení bez učitele* (unsupervised learning).

Při učení bez učitele neposkytujeme síti očekávaný výstup a síť si sama data třídí na základě nejrůznějších vzorů, které v datech najde. Co se týče učení s učitelem, síť dostane při trénování k datům i očekávaný výstup a na základě vyhodnocení správnosti je možné vypočítat chybu sítě

a váhy adaptovat, aby v další iteraci učení bylo vyhodnocení přesnější. Pro výpočet takovéto korekce se využívají nejrůznější algoritmy jako např. *backpropagation*.

3.2.1 Backpropagation

Účelem algoritmu backpropagation (zpětného šíření) je upravit synaptické váhy neuronů tak, aby měla síť co nejlepší výsledky. Algoritmus popisuje proces trénování (učení) sítě. Výsledkem je neuronová síť s minimalizovanou chybovostí. Tento algoritmus může být použit za předpokladu, že je k dispozici množina trénovacích dat a půjde o takzvané učení s učitelem. Jinými slovy, vstupní data musí být popsána požadovaným výstupem. Přenosové funkce neuronu musí být spojité, monotónně neklesající a diferencovatelné. Algoritmus lze rozdělit do tří hlavních kroků:

1. Dopředné šíření vstupního signálu
2. Zpětné šíření chyby
3. Adaptace vah neuronů

Tyto části se opakují do doby, kdy je chyba sítě tak malá, že síť můžeme považovat za naučenou. Chybu sítě můžeme vypočítat podle následujícího vzorce

$$E = \sum_{i=0}^{q} \frac{1}{2} \sum_{k=0}^{n} (y_{i,k} - t_{i,k})^2, \tag{7}$$

kde q je počet prvků trénovací sady, n je celkový počet výstupů, $y_{i,k}$ a $t_{i,k}$ jsou výstupy naučené a požadované. Na prvním místě při dopředném šíření signálu obdrží každý neuron vstupní signál a na jeho základě vypočítá svou hodnotu, kterou předá neuronům vyšší úrovně. U výstupní vrstvy je výsledná hodnota neuronů celkový výsledek klasfickace [9].

Posléze je pro každý prvek z trénovací množiny porovnána vypočtená a požadovaná hodnota a na základě rozdílu se vypočítává faktor, který obsahuje informaci o chybě šířené v síti směrem dolů. Adaptace vah vstupů neuronů se odvíjí od celkového chybového faktoru, který je definován vztahem

$$\delta_k^n = (t_k - y_k) f^{'}(w_{0,k}^n + \sum_{i=1}^{m} y_i^{n-1} w_{i,k}^n), \tag{8}$$

kde y_k je vypočtený výstup sítě a t_k chtěný výstup zadefinovaný v trénovací množině. $f^{'}$ je derivace aktivační funkce a m je celkový počet výstupů neuronů z nižší (předchozí) vrstvy [9]. Faktor chyby k-tého neuronu n-té skryté vrstvy můžeme zjistit z následujícího vztahu

$$\delta_k^n = \sum_{i=1}^{q} (\delta_i^{n+1} w_{k,i}^{n+1}) f'(w_{0,k}^n + \sum_{i=1}^{m} y_i^{n-1} w_{i,k}^n), \tag{9}$$

kde δ_i^{n+1} je faktor chyby i-tého neuronu vyšší vrstvy, q je celkový počet neuronů ve vyšší vrstvě a $w_{k,i}^{n+1}$ je váha mezi k-tým neuronem n-té vrstvy a i-tým neuronem vyšší vrstvy. f' je opět derivace aktivační funkce a m je počet výstupů v nižší vrstvě. V poslední řadě každé iterace algoritmu se upraví váhy neuronů. Hodnota změny biasu k-tého neuronu v n-té vrstvě je dána vztahem

$$w_{0,k}^n = k\delta_k^n, \tag{10}$$

kde k je koeficient učení, který zpravidla zpomaluje učení, aby adaptace vah byla přesnější. δ_k^n je faktor chyby neuronu [9].

3.3 Hluboké neuronové sítě

Myšlenky hlubokých neuronových sítí přišly poměrně nedávno, kdysi neuronové sítě obsahovaly pouze pár vrstev (typicky jednu až dvě). Bylo to proto, že se taková síť učila velice těžko, jelikož algoritmus backpropagation byl málo účinný v té chvíli, kdy se gradient chyby pohyboval okolo nízkých hodnot. Mnohovrstvé neuronové sítě se často potýkají i s problémem přeučení, a to v případech, kdy trénovací množina dat je malá a iterací učení je přespříliš, poté se síť naučí šum nebo jiné nedůležité příznaky z množiny. Přeučení má za důsledek špatný celkový výkon sítě. Nejtypičtějším příkladem těchto sítí je konvoluční neuronová síť [8].

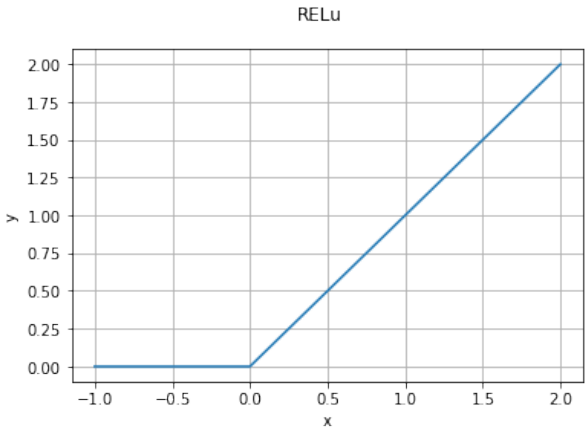
3.3.1 Techniky v hlubokých sítích

Důležitým krokem k správnému využití a naučení vícevrstvých neuronových sítí byla aplikace jiné aktivační funkce - ReLU (Rectified Linear Unit). Její hlavní vyhodou je, že při vysokých hodnotách x má funkce konstantní gradient. To znamenalo, že se takovéto sítě začaly učit rychleji, a to i při větším množství skrytých vrstev. Další důležitou výhodou je menší výpočetní náročnost oproti dříve užívanými exponenciálními funkcemi [8].

$$f(x) = \max 0, x \tag{11}$$

Dále se zjistilo, že je při učení výhodné vypnout občas některé neurony, této metodě se odborně říká *DropOut*. Jejím cílem je zabránit přeučení podobným způsobem, jakým funguje lidský mozek. Občas se může stát, že některé neurony odumírají, nebo jsou dočasně nedostupné. Tato metoda může být aplikována téměř na všechny vrstvy hluboké neuronové sítě od plně

propojených, až po konvoluční vrstvy. Když je metoda *DropOut* použita při učení sítě, tak by se při evaluaci v síti již objevovat neměla.



Obrázek 7: Průběh aktivační funkce ReLU

4 Konvoluční neuronové sítě

Pro klasifikaci a detekci objektů pomocí výpočetní techniky existuje dnes velká škála metod. Většina z nich má své pro a proti. V dnešní době, která je prosetá umělou inteligencí, se velikým trendem v rozpoznávání objektů z obrazu a mračna bodů staly konvoluční neuronové sítě.

Obecně je rozpoznávání objektů či barev pro počítače velmi obtížné, jelikož obrázky, videa nebo mračna bodů jsou pouze bezkontextové informace. Například bitmapový obrázek $32\ px\ \times\ 32\ px$ obsahuje informace o dílčích 1024 pixelech. Pro počítače to zkrátka nic víc neznamená, ale naopak pro člověka, po správnem vykreslení obrázku, může obraz představovat kupříkladu červené jablko. Proto je rozpoznávání objektů pomocí výpočetní techniky určitá výzva. Dnes užívané metody se stále zdokonalují, jelikož je pro nás čím dál tím více žádoucí, aby nám se smyslem vidění mohl počítač pomoci. Pokud budeme mít dostatečně dobré metody rozpoznání chodců na přechodu, počítač v autě zareaguje mnohonásobně rychleji a výrazně se sníží nehody na silnicích.

Nejčastěji probíhá detekce objektů tak, že v obraze hledáme různé barvy, hrany, nebo také souvislé plochy. Těmto sledovaným entitám říkáme příznaky a z nich bychom měli, pokud možno, objekt rozpoznat. Ne vždy je ale pro nás žádoucí sledovat veškeré příznaky, jelikož některé z nich můžou být pro daný případ rozpoznání irelevantní. Budeme-li chtít rozpoznávat auta, tak by nás jejich barva zajímat neměla. Často by nás naopak měly zajímat spíše malé, jedinečné tvary aut. Mezi další problémy, které se mohou vyskytnout, patří zejména špatná kvalita obrazu - obraz je plný šumu nebo rozostřený [8].

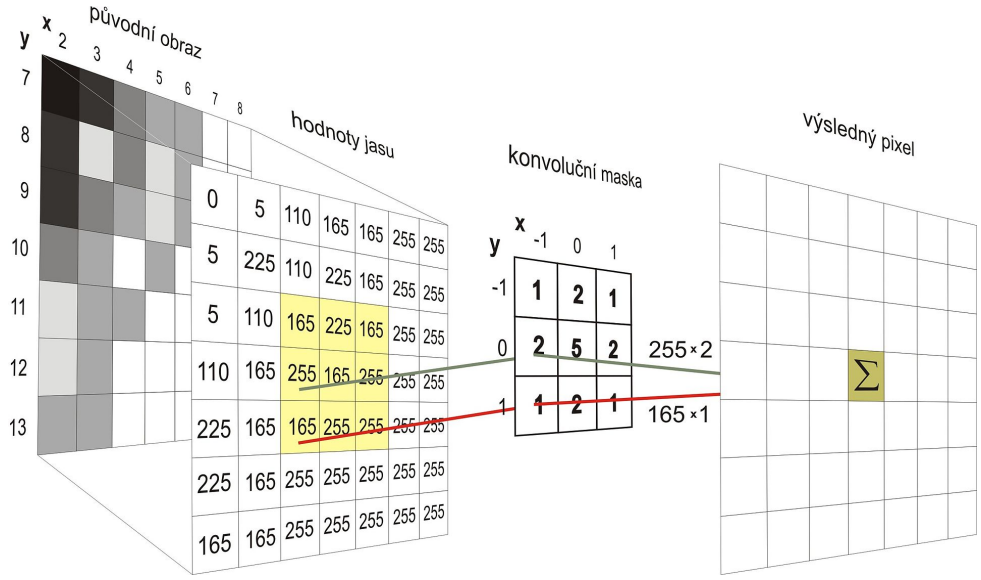
Konvoluční neuronové sítě se od neuronových sítí liší především tím, že vstupem nejsou jednotlivé příznaky, ale celý obraz. Síť si pomocí konvolučních vrstev dokáže důležité prvky obrazu vyextrahovat sama. Druhým důležitým rozdílem je, že konvoluční neuronové sítě používají techniky sdílení vah, kde více neuronů sdílejí tutéž váhu. To ve výsledku znamená, že pokud se neuron naučí rozpoznávat kupříkladu automobil a poté bude zaslán na vstup sítě zmíněný automobil pootočený, pravděpodobně se nedostane na stejný neuron. Může se avšak dostat k neuronu, se kterým sdílí ten původní svou váhu [8].

4.1 Konvoluce

Konvoluce je matematický operátor zpracovávající dvě funkce. Dnes se často používá v počítačové grafice, a to zejména v algoritmech pro zpracování 2D obrazu. Je definována vztahem

$$(f * g)(a) = \int_{-\infty}^{\infty} f(x)g(a - x)dx \tag{12}$$

kde $g(a)$ je konvoluční jádro, hodnota funkce f v bodě x je integrál součinu ze součinu funkce f s otočenou funkcí g. Při konvoluci ve zpracování obrazu je funkce f vstupní obrázek a funkce g konvoluční filtr.[8]



Obrázek 8: Příklad diskrétní konvoluce [10]

4.2 Konvoluční vrstva

Konvoluční vrstva navazuje na vstupní nebo pooling (subsamplingovou) vrstvu, je složena z map příznaků. Hlavní idea příznakových máp tkví v získání mnoho menších obrazů z předchozí vrstvy nebo vstupu. Tím uvažujeme například vstupní obrázek o rozměrech 32×32 pixelů a libovolný podobrázek o velikosti například 8×8 pixelů - tyto všechny derivované obrazy tvoří příznakovou mapu. Každý prvek z mapy je vstupem jednoho neuronu, ale všechny spolu sdílejí váhy. Tudíž posune-li se lokální oblast v obraze, pravděpodobně se výstup posune pouze o několik neuronů [11].

4.3 Subsamplingová (pooling) vrstva

Co se týče subsamplingové vrstvy, ta následuje vždy po konvoluční vrstvě, počet příznakových map se shoduje s počtem v předchozí vrstvě. Každý jeden neuron z příznakové mapy konvoluční vrstvy je propojen s jedním neuronem vrstvy subsamplingové. Účelem této vrstvy je snížit citlivost výstupu na otočení a jiné deformace obrazu. Důsledkem této operace je snížení počtu příznaků na základě hledání maxima (max pooling), nebo průměrné hodnoty (average pooling) v lokální oblasti mapy. Pokud tedy v praxi aplikujeme operaci max pooling s maticí kernelu typu 2×2, zmenšíme velikost příznakové mapy dvakrát [11].

4.4 Struktura CNN

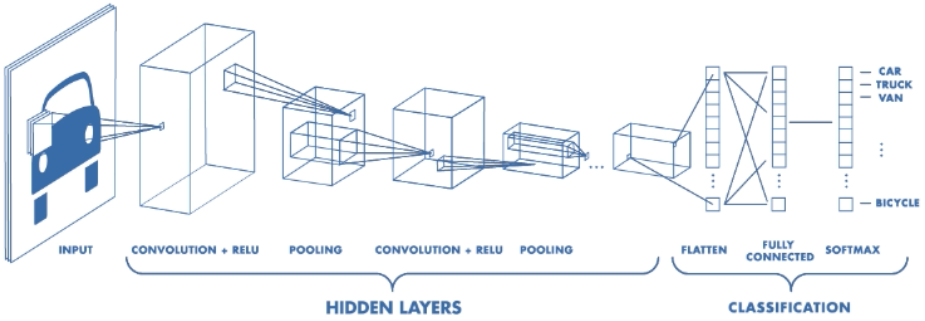
Konvoluční neuronové sítě mají za příklad sítě typu Neocognitron - síť navržena japonským vědcem Kunihiko Fukushimou, původně byla navržena pro rozpoznávání rukou psaného textu a jiných vzorů. Jak již bylo zmíněno, na vstup sítě je zasílán celý obraz, obdobně jak je tomu

tak u lidského mozku (vstupem u lidských neuronů je myšlena sítnice). Pokud je ovšem vstupem více vstupních map, (například kanál RGB) je před vykonáním operace první vrstvy, provedena sumace dle vztahu [8]

$$x_j^l = f(\sum_{e \in V_j} x_i^{l-1} * k_{ij}^l + bias_j^l) \tag{13}$$

kde x_j^l je výstup j-té mapy příznaků, v l-té vrstvě, f je aktivační funkce, V je množina vstupních map, x_i^{l-1} je i-tá příznaková mapa v $l-1$ vrstvě, k je konvoluční kernel patřící příslušné vstupní mapě dané vrstvy a *bias* je práh příslušné mapy dané vrstvy.

Síť je navržena tak, aby v jednotlivých vrstvách detekovaly jinak komplexní příznaky. V nízkých vrstvách se detekují jednoduché tvary, jako například přímky, které se ve vyšších vrstvách různě kombinují, až nakonec v poslední vrstvě vznikne obraz. Typicky obsahují konvoluční neuronové sítě několik konvolučních vrstev, které jsou následovány vrstvy subsamplingovými. Tato síť je poté napojena na plně propojenou neuronovou síť, která dále příznaky zpracovává a tvoří výstup sítě. Typická struktura konvoluční neuronové sítě je uvedena na obrázku (9) [8].



Obrázek 9: Typická struktura konvoluční neuronové sítě [12]

5 Dostupné práce založené na CNN

Přístupy aplikované pro segmentaci a klasifikaci mračna bodů před konvolučními neuronovými sítěmi už dnes nejsou brány na příliš velikou váhu, protože spočívaly především na člověkem obtížně získávanými vlastnostmi jako třeba povrchové normály nebo generované deskriptory. Dalším problémem je zdlouhavý proces nastavování prahu shlukových algoritmů. Tyto přístupy bohužel nejsou dostatečně dobré a rychlé, aby byly použitelné pro segmentaci a klasifikaci v produkčním prostředí. Z těchto důvodů se v kapitole budeme zabývat pouze přístupy, které jsou založené na konvolučních neuronových sítích.

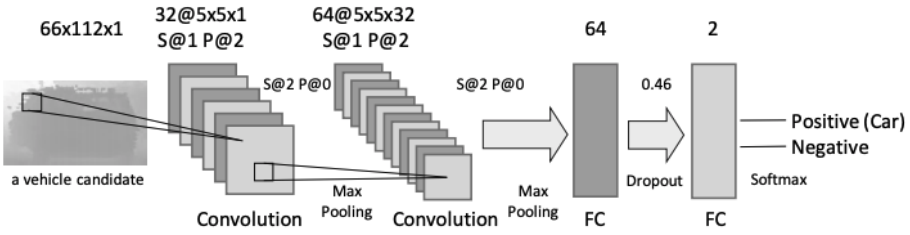
5.1 DepthCN

Tato vědecká práce popisuje a řeší problematiku detekce a rozpoznávání vozidel pomocí hluboké konvoluční neuronové sítě s aplikací v pokročilých asistenčních řídících systémech a v autonomním řízení, tím pádem je důraz kladen především na rychlost a správnost. Vstupem celého algoritmu je mračno bodů ve formátu PCD, model sítě je odvozen z *ConvNet*, programovacím jazykem celého řešení je MATLAB. Trénovací a testovací množina neuronové sítě je převzata z KITTI [2]. Výstupem neuronové sítě je hloubková mapa s rámečky ohraňující detekované automobily.

Tento algoritmus začíná odstraněním bodů země z mračna, poté následuje segmentace PC (mračna bodů). V dalším kroku vzniká hloubková mapa projekcí 3D dat na 2D plochu pomocí Delaunayovy triangulační metody. Z hloubkové mapy se vyextrahují kandidáti pomocí shlukovací metody DBSCAN a v posledním kroku probíhá klasifikace, zda je hledaný objekt automobil, nebo není. Za účelem snížení výpočetního času a odstranění nežádoucího „šumu“ v mračnu bodů se odebírají body, které jsou od sebe mírně invariantní v ose Z [13] .

5.1.1 Struktura sítě

Vstupem je hloubková mapa o velikosti 66×112. V první vrstvě se provádí konvoluce s kernelem o velikosti $5 \times 5 \times 1$ s krokem 1 a odsazením 2. Následuje operace *max-pooling*, poté druhá konvoluce s kernelem velkým 5×5, s krokem 1 a odsazením 2. Následuje opět *max-pooling* do plně propojené vrstvy o 64 neuronech na kterou je aplikována vrstva *DropOut* s koeficientem $0,46$, což znamená, že 46% neuronů bude náhodně vyřazeno, aby se zabránilo přeučení. Poslední vrstva se skládá ze dvou neuronů, kde první indikuje rozpoznané auto a druhý negativní výstup. Výstup je získáván aktivační funkcí softmax [13].



Obrázek 10: Struktura sítě ConvNet [13]

5.1.2 Dosažené výsledky

Oproti jiným řešením měří *DepthCN* své výsledky metrikou přenosti. Přesnost je dána vztahem

$$Pr = \frac{|A_x \cap A_y|}{|A_x|}, \tag{14}$$

kde A_x je množina predikovaných bodů a A_y je množina bodů, které doopravdy náleží danému objektu.

Množina dat	**Přesnost**
Trénovací	96,02 %
Testovací	91,93 %

Tabulka 2: Dosažené výsledky DepthCN [13]

5.2 SqueezeSeg

Mezi další důležité práce v oblasti segmentace objektů z LiDARových dat řadíme *SqueezeSeg*. Jde o projekt, který zajišťuje segmentaci a klasifikaci objektů vyskytujících se nejčastěji v bezprostřední blízkosti cest. Hlavní výhodou tohoto algoritmu je jeho velmi vysoká rychlost a stabilita. Jeden snímek dokáže zpracovat za $8,7 \pm 0,5$ ms.

Vstupními daty algoritmu je mračno bodů v kartézském souřadnicovém systému, síť vychází z modelu *SqueezeNet*. Celá práce je naprogramována v jazyce Python s pomocí knihovny *TensorFlow*. Data pro trénování a testování pochází jednak z KITTI [2], ale také z vlastní simulace LiDARu ve hře *GTA V*. Výstupem algoritmu je LiDARová maska s vyznačenými segmentovanými automobily, cyklisty a chodci.

Předchozí přístupy často závisely, stejně jako u *DepthCN*, na lidmi ručně laděnými algoritmy typu *RANSAC*, které zajišťovaly například odstranění bodů země, které v této sémantické segmentaci považujeme za šum. Nicméně například algoritmus *RANSAC* nedokáže zajistit stabilní

výsledky, jelikož je do jisté míry náhodný. Navíc jeho použití je časově nákladné a nevhodné k *real-time* segmentaci [14].

Algoritmus je postaven na přístupu konovoluční neuronové sítě a podmíněného náhodného pole (CRF), které tvoří poslední vrstvu sítě jako rekurentní neuronová síť. Oba tyto přístupy již byly úspěšně pro segmentaci objektů ve 2D obraze použity. Zatímco CNN zastává funkci predikce pravděpodobnostní mapy, CRF je použito k obnovení detailů mračna bodů [14].

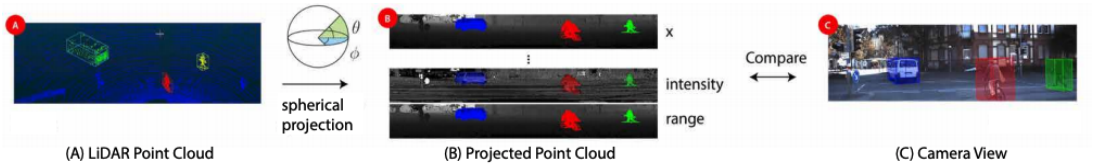
Vyhodnocení jednoho snímku začíná sférickou projekcí mračna bodů na 2D plochu. Obvyklým vstupem konvoluční neuronové sítě je matice o velikosti $H \times W \times C$, kde H a W jsou výška a šířka obrazu a C je počet vlastností - typicky u barevných obrazů jsou to kanály modré, červené a zelené barvy. LiDAR těmito vlastnostmi nedisponuje, tudíž se nabízí použití souřadnic trojic $\{x, y, z\}$, které jsou běžně používány k opisu bodů v mračnu [14].

Pro získání kompaktní reprezentace mračna v 2D obraze je nutno aplikovat na každý bod následující výpočty a získat tak úhel azimutu a elevace [14].

$$\theta = \arcsin \frac{z}{\sqrt{x^2 + y^2 + z^2}}, \overline{\theta} = \lfloor \theta / \Delta\theta \rfloor \tag{15}$$

$$\varphi = \arcsin \frac{y}{\sqrt{x^2 + y^2}}, \overline{\varphi} = \lfloor \varphi / \Delta\varphi \rfloor \tag{16}$$

kde θ a φ jsou úhly azimutu a zenitu. $\Delta\theta$ a $\Delta\varphi$ jsou rozlišení pro diskretizaci a udávají pozici bodu na 2D sférické mřížce. Jelikož jsou data získávána ze zařízení Velodyne HDL-64E, které disponuje 64 vertikálními kanály, výška H bude 64. Nejdůležitější částí mračna je oblast před vozidlem s LiDARem, proto je bráno v potaz pouze horizontální oblast 90°, která bude rozdělena do 512 mřížek. Počet mřížek koresponduje s šířkou výsledného obrazu, tudíž můžeme říci, že $W = 512$. Pro C evidujeme kromě souřadnic x, y, z i *intenzitu* a *vzdálenost*, tím pádem je $C = 5$ [14].



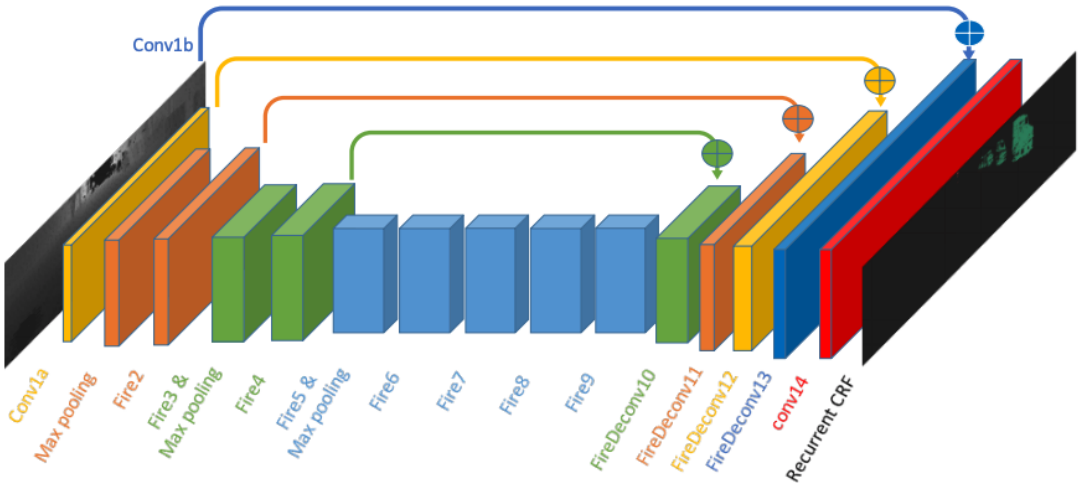
Obrázek 11: Sférická projekce na 2D mřížku. *Obrázek A* zobrazuje mračno bodů, *obrázek B* demonstruje projekci a *obrázek C* je korespondující obraz ze zkalibrované kamery [14]

5.2.1 Struktura sítě

Model sítě je založen na *SqueezeNet*. Vstupem je matice o velikosti $64 \times 512 \times 5$. *SqueezeNet* používá *max-pooling* ve směru šířky i výšky, ale tento přístup není v tomto případě vhodný, protože výška vstupního obrazu je mnohem menší než šířka. Z tohoto důvodu se zde provádí operace *max-pooling* u vrstev od *conv1a fire9* pouze do šířky [15].

Zmíněná síť, na které je *SqueezeSeg* založena, obsahuje tzv. *fire* konvoluční a dekonvoluční moduly, které markantně snižují počet parametrů a výpočetní náročnost modelu. Jejich struktura je podrobně popsána v další kapitole a jejich architektura je zobrazena na obrázku (18).

V této segmentační metodě mají predikované objekty tendenci mít nepřesné detaily hran, to je zapříčiněno ztrátou patřičných detailů podvzorkováním. Kvůli tomu se autoři rozhodli použít metodu *Conditional Random Field* jako poslední vrstvu sítě, která výslednou masku s predikovanými objekty zkvalitní [15].



Obrázek 12: Model sítě SqueezeSeg [14]

5.2.2 Data

Metoda *SqueezeSeg* nabývá na zajímavosti také tím, že data pro vstup sítě nepochází pouze z reálného světa, ale i ze simulace LiDARu pomocí metody *ray casting* ve hře GTA V, kde na střechu auta připevnili virtuální LiDARový senzor a nechali auto řídit autonomně. Každému bodu zachycenému snímačem se poté automaticky přiřadí štítek z kategorií *automobil, ostatní*. Je důležité také zmínit, že cyklisté ve hře nemají svou kategorii a jsou považování za chodce, zřetel je brán pouze na auta. Touto metodou bylo získáno 8 585 snímků s korespondujícími záznamy ze hry [15].

Bohužel, tato data neobsahují šum, kterým disponují data KITTI [2], což znamená, že při učení sítě z tohoto vzorku se pravděpodobně sníží úspěšnost rozpoznání objektů na reálných datech. Z tohoto důvodu se autoři rozhodli šum do syntetických snímků dodat. Autoři vycházeli z dat KITTI [2], ze kterých vyextrahovali části obrazu, kde se rušivé části nejčastěji nachází a rovnoměrně je do syntetických snímků přidali.

5.2.3 Dosažené výsledky

SqueezeSeg používá pro měření výsledků primárně metriku *Intersection over Union (IoU)*, která se řadí mezi přísnější. Jejím výsledkem je číslo v intervalu $\langle 0, 1 \rangle$, kde větší hodnoty znamenají lepší výsledek. Je definována dle vztahu

$$IoU = \frac{|A_x \cap A_y|}{|A_x \cup A_y|}, \tag{17}$$

kde A_x je množina bodů, náležící predikovanému objektu a A_y je množina všech bodů, která náleží objektu v obraze.

Původ dat	**IoU**
KITTI [2]	57,1 %
GTA	29,0 %
KITTI [2] + GTA	66,0 %

Tabulka 3: Dosažené výsledky SqueezeSeg s CRF vrstvou [14]

5.3 PointSeg

Další z vybraných prací zaměřených na segmentaci a klasifikaci mračna bodů je *PointSeg*. Je založena na práci kalifornské univerzity Berkeley - *SqueezeSeg*, kterou jsme již rozebrali. Hlavním cílem autorů *PointSeg* je vytvořit dostatečně rychlou metodu segmentace mračna bodů, aby byla použitelná v robotice a autonomně řízených autech. Jelikož jsou mračna bodů z LiDARu známá tím, že jeden snímek obsahuje množství bodů v řádech statisíců, je výzva navrhnout takový algoritmus, aby byl použitelný pro segmentaci v reálném čase.

Vstupními daty je mračno bodů v kartézském souřadnicovém systému. Celý model sítě je založený na SqueezeNet [15], množina trénovacích a testovacích dat pochází z KITTI [2] databáze, která obsahuje celkem 7481 LiDARových snímků. Výstupem celého procesu rozpoznání objektů je opět mračno bodů, tentokrát ale s vyznačenými automobily, cyklisty a chodci.

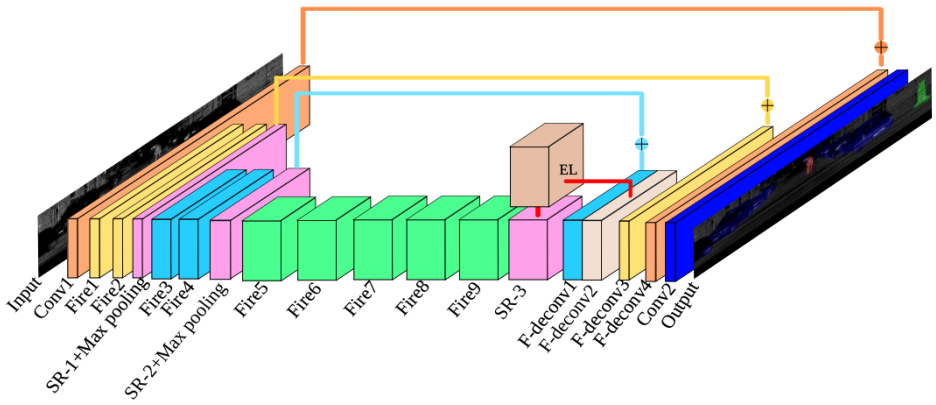
Aby byly splněny požadavky rychlosti a přesnosti, byla vybrána výše zmíněná síť *SqueezeNet* [15], která dosahuje velice podobných výsledků jako síť *AlexNet* a zároveň je model velikostně až 50krát menší. Důsledkem je mnohem rychlejší zpracování vstupu neuronové sítě nebo také rychlejší učení.

Informace o bodech z PC jsou často ukládány jako množina bodů $P = \{x, y, z\}, P \in C$, kde P je jeden bod zastoupen souřadnicemi x, y, z z množiny mračna C. Obvykle se v P vyskytuje i informace o intenzitě vyslaného paprsku. Navíc, pokud je LiDAR zkalibrován s kamerami, jednoduše můžeme získat i RGB informace jednotlivých bodů.

Prvním krokem je provést sférickou projekci bodů na 2D plochu o velikosti 64×512. Ta probíhá stejně jako u metody *SqueezeSeg* zmíněné výše. Pro každý jeden snímek se bude uchovávat informace o souřadnicích $x, y, z, intenzitě$ a navíc i vzdálenosti l. Díky této projekci se výrazně zmenší velikost vstupu neuronové sítě od případů, kde je vstupem celé mračno bodů [16].

5.3.1 Struktura sítě

Oproti běžným konvolučním neuronovým sítím obsahuje tento model i takzvané *Fire* konvoluční a dekonvoluční vrstvy. Podrobněji budou rozebrány v následující kapitole. Jejich architektura je zobrazena na obrázku (18). Hlavním důvodem použití těchto vrstev je, jak již bylo zmíněno, snížení počtu parametrů modelu a zvýšení rychlosti učení [15].



Obrázek 13: Struktura sítě PointSeg [16]

5.3.2 Dosažené výsledky

Práce *PointSeg* měří své výsledky pomocí metrik *IoU*, *recall* a *precision*. Metrika *IoU* již byla představena v předchozí práci *SqueezeSeg*, je definována vztahem z rovnice (17). Přesnost (precision) Pr je definována dle rovnice (14). V poslední řadě metrika *recall* R je dána vztahem

$$R = \frac{|A_x \cap A_y|}{|A_y|}, \tag{18}$$

kde A_x je množina bodů predikovaného objektu a množina A_y obsahuje ty body, které doopravdy náleží objektu v obraze.

Objekt	Precision	Recall	IoU
Automobil	74,8 %	92,3 %	67,4 %
Chodec	41,4 %	29,3 %	19,2 %
Cyklista	41,4 %	59,7 %	32,7 %

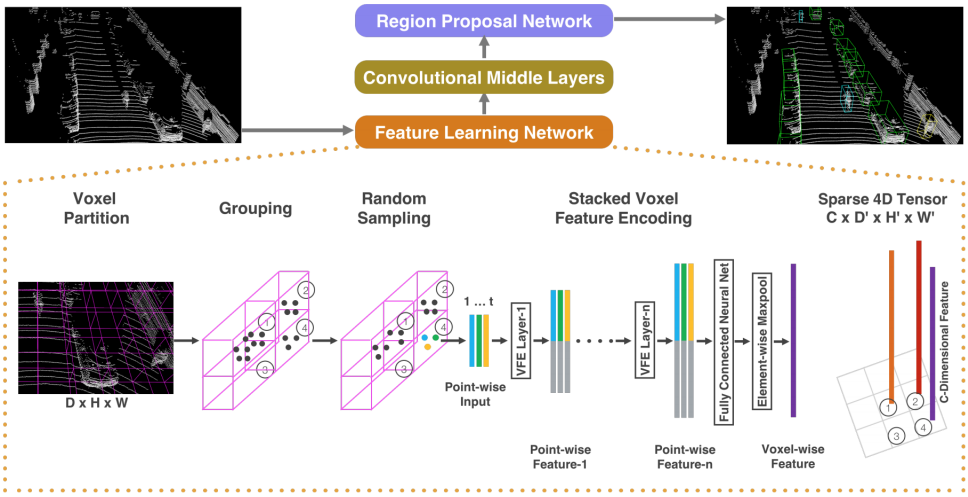
Tabulka 4: Dosažené výsledky PointSeg [16]

5.4 VoxelNet

Oproti ostatním, níže zmíněným, algoritmům je *VoxelNet* výjimečný tím, že vstupní mračno bodů nepřevádí na hloubkovou mapu, ale provádí segmentaci na samotném mračnu. Výhodou takového přístupu je, že data není nutno předzpracovávat. Nicméně hlavní nevýhodou této metody je její rychlost, která se pohybuje přibližně okolo 250 ms za jeden zpracovaný snímek.

Vstupními daty této práce výzkumných pracovníků firmy *Apple* je mračno bodů v kartézském souřadnicovém systému. Celá síť byla učena a testována na datech z *KITTI* [2]. Výsledkem neuronové sítě je pak mračno bodů s 3D boxy označující segmentované automobily, cyklisty a chodce.

Celý algoritmus je celkově rozdělen do tří částí, kde první část je označována jako Feature Learning Network (FLN), druhá se skládá z konvolučních mezivrstev a třetí část je nazvána Region Proposal Network (RPN) [17].



Obrázek 14: Struktura algoritmu VoxelNet [17]

5.4.1 Feature Learning Network

Prvním krokem je rovnoměrné rozdělení mračna bodů do takzvaných *voxelů* (jde o analogii k pixelu v 2D, narozdíl od pixelu je třídimenzionální). Poté se seskupí body, které náleží příslušným voxelům a vzniknou skupiny, které mohou být i prázdné. Dalším krokem je náhodné vzorkování,

které obnáší selekci skupin voxelů, které obsahují více bodů, než číslo T, které bylo náhodně zvoleno. Toto vzorkování má dva hlavní účely - zmenšení výpočetního času, snížení nerovnováhy mezi voxely [17].

Poté následuje zakódování vlastností voxelů. Mějme neprázdný voxel

$$V = \{p_i = [x_i, y_i, z_i, r_i]^T \in \mathbb{R}^4\}_{i=1...t}, \tag{19}$$

kde p je reprezentace i-tého LiDARového bodu, x, y, z jsou korespondující souřadnice, r je intenzita reflektance a t je počet bodů ve voxelu. Nejprve je vypočítán centroid voxelu $C = c_x, c_y, c_z$ pro všechny souřadnice x, y, z jako

$$c_a = \sum_{i=1}^{t} \frac{1}{t}(a_1 + a_2 + ... + a_t), \tag{20}$$

kde t je počet bodů voxelu, a je korespondující souřadnice. Poté se rozšíří množina bodů ve voxelu o ofset od centroidu

$$V_{in} = \{p_i = [x_i, y_i, z_i, r_i, x_i - c_x, y_i - c_y, z_i - c_z]^T \in \mathbb{R}^7\}_{i=1...t} \tag{21}$$

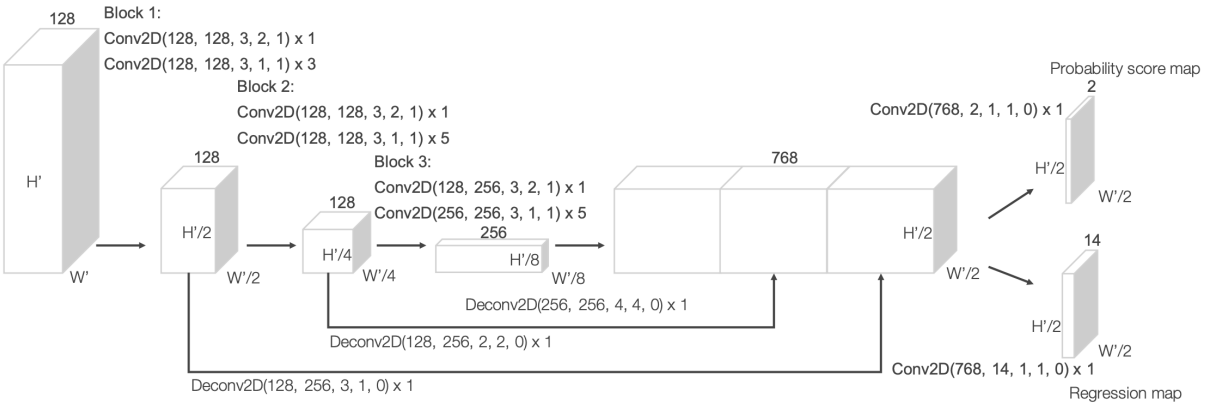
Poté je každý bod p_i transformován do prostoru vlastností, kde jsou dále agregovány tvary povrchu obsažené ve voxelu. Díky vyřazení prázdných voxelů jich zůstane přibližně 10 %, a to je klíčové pro optimalizaci výkonu celého procesu [17].

5.4.2 Konvoluční mezivrstvy

Do procesu jsou přidány třídimenzionální konvoluční vrstvy následované normalizací a aktivační funkcí ReLU. Výstupem této, a zároveň vstupem třetí části, je mapa vlastností voxelů [17].

5.4.3 Region Proposal Network

Tento druh neuronové sítě slouží k označení regionů v datech. V uvedeném případě má výstupy dva: pravděpodobností výsledná mapa a regresní mapa. Obvykle je výstupem takovéto sítě několik ohraničujících rámečků, které klasifikátor dále vyhodnocuje [17].



Obrázek 15: Struktura sítě Region Proposal Network (VoxelNet) [17]

5.4.4 Dosažené výsledky

Metoda *VoxelNet* patří mezi jednu z nejpřesnějších ve své kategorii, navíc nevyžaduje žádné předzpracování dat. V již dříve představené metrice IoU dosáhla 66,7 %, což ji činí velikým konkurentem např. metody *SqueezeSeg* [17].

6 Praktická část - rozpoznání objektů z LiDARových dat

Jak již bylo zmíněno v předchozích kapitolách, je možné dosáhnout dobrých výsledků v detekci objektů z LiDARových dat transformací na 2D mřížku, která se stane vstupem pro konvoluční neuronovou síť. Pro účely této práce jsme vybrali model SqueezeSeg se vstupní vrstvou 64×512 z důvodu malé velikosti modelu sítě a rychlosti zpracování tak vysoké, že je vhodná i pro zpracování v reálném čase.

Jako výchozí zdroj dat jsme použili data KITTI [2], jelikož jsou precizně označkována a četnost dat je této práci dostačující. Každý snímek množiny má i korespondující soubor obsahující data o objektech, které se v nich nachází. V poslední řadě se u dat vyskytují také kalibrační soubory, kterými můžeme zkalibrovat záznamy z kamer a LiDARových snímků.

K implementaci algoritmů předzpracování dat a konvoluční neuronové sítě byl použit programovací jazyk *Python*, který vévodí datové analytice a neuronovým sítím za pomocí rozmanité škály knihoven. K samotné implementaci konvoluční neuronové sítě používáme knihovnu *TensorFlow*, která poskytuje vysokoúrovňové rozhraní pro trénování sítí a následnou evaluaci.

Další důležitou součástí trénování neuronové sítě je použitý hardware. Jelikož je učení výpočetně vysoce náročné, je vysoce doporučené jej paralelizovat pomocí technologie CUDA, kterou využívají grafické karty NVIDIA. Učení neuronových sítí, zejména pak konvolučních, na zařízení s GPU dokáže zvýšit výkon v řadách stanásobků [18].

6.1 TensorFlow Estimator

Celá níže uvedená síť je naprogramována pomocí rozhraní modulu *TensorFlow Estimator* (TFE). Hlavními výhodami použití je zjednodušení implementace (automatické vytvoření výpočetního grafu), podpora distribuovaného učení a podpora modulu *TensorBoard*, který slouží jako vizualizační nástroj při učení s velmi jednoduchým a účinným rozhraním [18].

Jednoduchost a přímočarost implementace můžeme vidět na ukázce zdrojového kódu (1), kde v prvé řadě vytváříme instanci klasifikátoru s námi specifikovaným modelem sítě, dále klasifikátor trénujeme pomocí metody klasifikátoru *train* ve smyčce dlouhé 2000 kroků. Nakonec natrénovaný klasifikátor otestujeme na datech, které jsme zatím modelu neposkytli, a ověříme správnost klasifikace nebo případně segmentace.

```
# vytvoření klasifikátoru
classifier = tf.estimator.Estimator(
       model_fn=squeeze_seg_fn, # funkce, kde je definována struktura sítě
       model_dir='./model/') # cesta ke složce, kde se bude trénovaný model
           ukládat

# funkce s trénovacími daty modelu sítě
train_input_fn = tf.estimator.inputs.numpy_input_fn(
```

```
        x={"x": train_data}, # vstupní trénovací data
        y=train_labels, # vstupní masky označkovaných dat
        batch_size=4, # počet prvků v jedné trénovací skupině
        num_epochs=32, # počet epoch trénovací skupiny
        shuffle=True)

# trénovaní našeho klasifikátoru se vstupními daty definovány v proměnné
    train_input_fn ve 2000 krocích
classifier.train(
        input_fn=train_input_fn,
        steps=2000)

# funkce s testovacími daty modelu sítě
eval_input_fn = tf.estimator.inputs.numpy_input_fn(
        x={"x": eval_data},
        y=eval_labels,
        num_epochs=1,
        shuffle=False)

# testovaní dat natrénovaným klasifikátorem
eval_results = classifier.evaluate(input_fn=eval_input_fn)
```

Výpis 1: Vytvoření TFE pro trénování a testování

Takto definovaný zdrojový kód, pomineme-li absenci importu knihoven, definice modelové funkce a metod načítaní trénovacích a testovacích dat, je plně dostačující k natrénování a otestování modelu. Mimo *TFE* existuje i jiné vysokoúrovňové rozhraní pro definici a estimaci modelu. Například jde o balíček *Keras*, který je nyní součástí knihovny *TF*.

6.2 Předzpracování dat

Klíčovou roli při zpracování dat je jejich příprava. Většinou zabere více času, než samotné zpracování. Často se jedná o redukci šumu, extrakci parametrů nebo také převedení dat do požadovaného formátu. Většinou je pro tyto techniky volen programovací jazyk Python, jelikož jsou pro něj dostupné *open-source* knihovny pro nejrůznější manipulaci s daty.

Knihovna	Popis
Pandas	Knihovna sloužící pro analýzu a manipulaci s daty
NumPy	Poskytuje rozhraní pro práci s vektory, maticemi a vícerozměrnými poli (tensory)
Matplotlib	Knihovna zajišťující vizualizaci dat pomocí grafů

Tabulka 5: Nejznámější knihovny jazyka Python ke zpracovávání a analyzování dat

Našim úkolem je převést vstupní mračno bodů na 2D mřížku o velikosti 64×512, která je vstupem konvoluční neuronové sítě, kterou představíme v následující kapitole. Jelikož v množině dat KITTI [2] jsou označkovány objekty pouze v kamerových záznamech, nebudeme potřebovat všechny body mračna. Konkrétně tedy body, které jsou v zorném poli řidiče. Jelikož zařízení Velodyne HDL-64E snímá vertikálně pouze $26,8°$, souřadnici z do následující podmínky pro selekci zahrnovat nebudeme. Aplikujme tedy následující podmínku na každý bod mračna.

$$P_{x,y,z,i} \in FoV \Leftrightarrow \operatorname{atan2} P_y, P_x > \frac{\pi}{4} \wedge \operatorname{atan2} P_y, P_x < -\frac{\pi}{4}, \tag{22}$$

kde $P_{x,y,z,i}$ je bod mračna reprezentován souřadnicemi P_x, P, y a P_z, FoV (angl. *Field of View*) je množina bodů výhledu řidiče dána horizontálními úhly od $-45°$ do $45°$.

Takto získané body z množiny FoV jsou téměř přípraveny ke sférické projekci na 2D mřížku. Body zatím neobsahují informace o vzdálenosti a o označkovaných objektech. Vzdálenost je dána vztahem $P_r = \sqrt{P_x^2 + P_y^2 + P_z^2}$, kde P_r je vzdálenost bodu od lidaru a $P_{x,y,z}$ jsou souřadnice bodu P.

Jak již dříve bylo zmíněno, tak informace o značkách objektů jsou v textovém souboru, kde jeden řádek představuje jednu značku. Informace v rámci jednoho řádku jsou odděleny mezerami a jejich struktura je následující:

1. **Objekt** - Jeden z následujících: *Car*, *Van*, *Truck*, *Pedestrian*, *Person_sitting*, *Cyclist*, *Tram*, *Misc* nebo *DontCare*.
2. **Opouští obraz** - Hodnota v intervalu $\langle 0, 1 \rangle$, kde hodnota určuje, do jaké míry je objekt mimo obraz.
3. **Viditelnost** - Celé číslo z množiny $0, 1, 2, 3$, kde 0 - plně viditelné, 1 - částečně zakryto, 2 - výrazně zakryto, 3 - není známo.
4. **Pozorovací úhel** - Nabývá hodnot od $-\pi$ do π.
5. **2D rámeček** - 4 souřadnice v obrázku definující hranice objektu.
6. **3D box** - 3 dimenze ohraničujícího boxu v pořadí výška, šířka, délka.
7. **3D umístění** - 3 souřadnice v prostoru a v pořadí x, y, z.

8. **Rotace Y** - Rotace objektu kolem osy y. Nabývá hodnot od $-\pi$ do π.

Pro naše účely jsou zajímavé segmenty **objekt**, **3D box**, **3D umístění**, **Rotace Y**. Nyní zjistíme, které souřadnice náleží příslušným objektům ze korespondujícího souboru, rozšíříme bod P o hodnotu, jakému objektu bod náleží. Pokud náleží autu, označíme jej 1, pokud cyklistovi, tak 2, jestliže je bod součástí chodce, nabyde hodnoty 3.

Tak, jak je definován 3D box a 3D umístění, nám nestačí, jelikož objekty jsou téměř vždy natočeny a kdybychom použili tyto hodnoty, znamenalo by to, že bychom objekty v mračnu často neoznačili celé, nebo chybně označili body, které objektu nenáleží. Musíme tedy použít rotační matici R_y

$$R_y(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}, \tag{23}$$

kde ϕ je rotační úhel (Rotace Y).

Po aplikaci rotační matice na objekt získáme box objektu, který ohraničuje objekt přesně. To, které body podléhají danému objektu, zjistíme pomocí vztahu

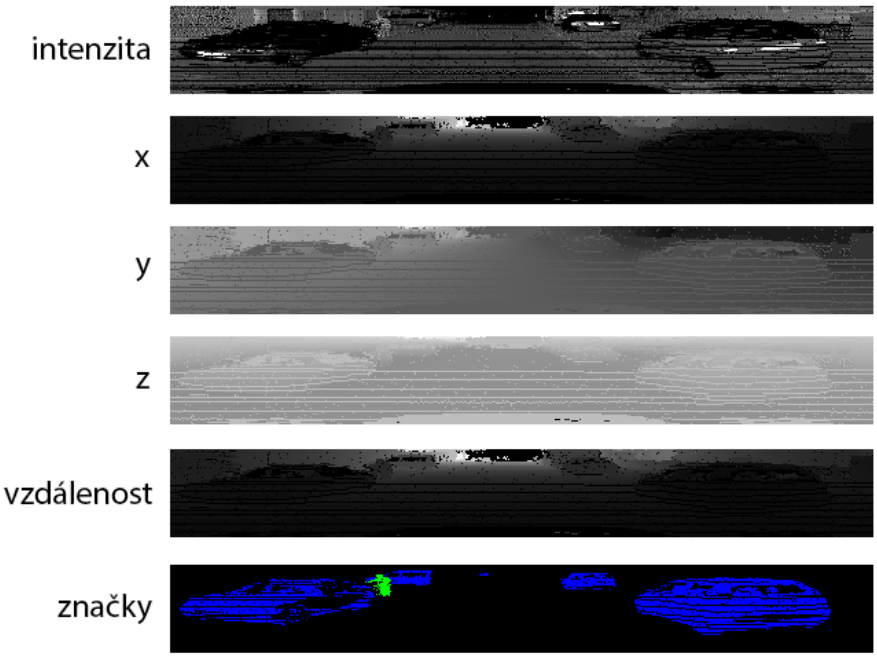
$$\begin{aligned} P_{x,y,z} \in B \Leftrightarrow (P_x >= B_{minX} \wedge P_x <= B_{maxX}) \wedge \\ (P_y >= B_{minY} \wedge P_y <= B_{maxY}) \wedge \\ (P_z >= B_{minZ} \wedge P_z <= B_{maxZ}) \end{aligned} \tag{24}$$

kde $P_{x,y,z}$ je bod mračna a B je ohraničující box objektu. Těmto bodům přiřadíme příslušnou značku $\{0, 1, 2, 3\}$ a poté následuje sférická projekce na mřížku. Abychom získali kompaktní, hustou reprezentaci bodů ve 2D prostoru, aplikujeme na každý bod následující vzorce sférické projekce

$$\begin{aligned} \theta = \arcsin \frac{z}{\sqrt{x^2 + y^2 + z^2}}, \widetilde{\theta} = \lfloor \theta / \Delta\theta \rfloor, \\ \phi = \arcsin \frac{y}{\sqrt{x^2 + y^2}}, \widetilde{\phi} = \lfloor \phi / \Delta\phi \rfloor, \end{aligned} \tag{25}$$

kde θ a ϕ jsou úhly azimutu a zenitu, $\Delta\theta$ a $\Delta\phi$ jsou diskrétní rozlišení výšky, resp. šířky obrazu. Souřadnice v mřížce jsou dány $\widetilde{\phi}$ a $\widetilde{\theta}$.

Takto připravená data jsou již vhodná jako vstup konvoluční neuronové sítě. V tuto chvíli máme převedeno 7481 LiDARových snímků, kde pro každý pixel z mřížky 64×512 sledujeme 6 vlastností C. Konkrétně tedy x, y, z, i, r, l, kde x, y, z jsou souřadnice bodu, i je intenzita paprsku, r je vzdálenost bodu a l je číslo objektu, kterému pixel náleží. Nenáleží-li žádnému sledovanému objektu, bude nabývat hodnoty 0. Ostatní hodnoty již byly vypsány výše v kapitole.

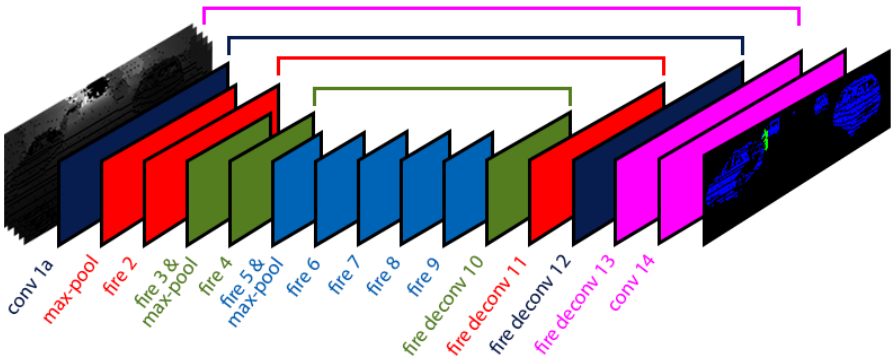


Obrázek 16: Vizualizace vstupních a očekávaných výstupních dat neuronové sítě s aplikací *min-max* normalizace. U značek je pro vyznačení automobilu použita barva modrá, pro cyklistu zelená a pro chodce barva červená.

Zpracovaná data nyní náhodně rozdělíme na trénovací a testovací množinu v poměru $3 : 1$, kde tři díly náleží trénovací a jeden díl testovací množině. Bylo by samozřejmě lepší, kdybychom množiny mohli rozdělit na dva stejně velké celky, ale vzhledem k relativně malému počtu dat by mohlo dříve docházet k přeučení, a navíc by klasifikátor ztrácel na přenosti.

6.3 Model CNN

Neexistuje žádný univerzální návod, jak model neuronové sítě navrhnout. Obecně však platí různé poučky, heuristické metody. Po prozkoumání dostupných modelů jsme zjistili, že výhodné je pro naše účely použít model založený na síti SqueezeNet, jelikož dosahuje dobrých výsledků a má relativně nízký počet parametrů sítě. Nicméně vzhledem k velikosti vstupních obrazů $64 \times 512 \times 5$, jej upravíme stejně jako ve *SqueezeSeg*, aby vzorkování probíhalo pouze do šířky.



Obrázek 17: Použitý model sítě založený na *SqueezeSeg* [14]

Celý model je graficky znázorněn na obrázku (17), nicméně nyní jej podrobně popíšeme. Vstupem první konvoluční vrstvy je výše zmíněný tensor o velikosti $64 \times 512 \times 5$, kde pro konvoluci je zvoleno 96 filtrů, kernel je velikosti $[3, 7]$ s krokem $[1, 2]$ a aktivační funkcí ReLU. Je následována vrstvou *max-pooling* s velikostí sdružovacího vzorku $[1, 2]$ a krokem $[1, 2]$.

Další částí sítě jsou dva tzv. *fire* moduly, jejichž architektura je zobrazena na obrázku (18), pro první konvoluční vrstvu modulu je použito 16 filtrů a pro vrstvy na druhé úrovni 64. Tyto moduly jsou následovány vrstvou *max-pooling* s krokem $[1, 2]$ a velikostí sdružovacího vzorku $[1, 2]$.

Následují čtyři *fire* konvoluční moduly, první dva se skupinou filtrů $[48, 192]$, druhé dva se skupinou filtrů $[64, 256]$. Po těchto vrstvách přicházejí na řadu dekonvoluční *upsampling* vrstvy. Konkrétně jde o *fire-dekonvoluční* moduly. Jejich struktura je spolu s klasickými *fire* moduly zobrazena na obrázku (18). V modelu sítě se vyskytují celkem čtyřikrát a každý modul je následován jednou spojovací vrstvou, která výstup modulu spojí s jednou z předchozích vrstev s korespondující velikostí.

Poslední vrstvou je konvoluční vrstva, kde počet filtrů je roven 4, velikost kernelu je $[1, 1]$ s krokem $[1, 1]$ a její výstup je aktivován funkcí ReLU. Z ní dokážeme, pomocí vyextrahování indexů největších hodnot všech os, získat mapu predikovaných objektů, která se používá pro výpočet metrik a určení správnosti klasifikátoru [14].

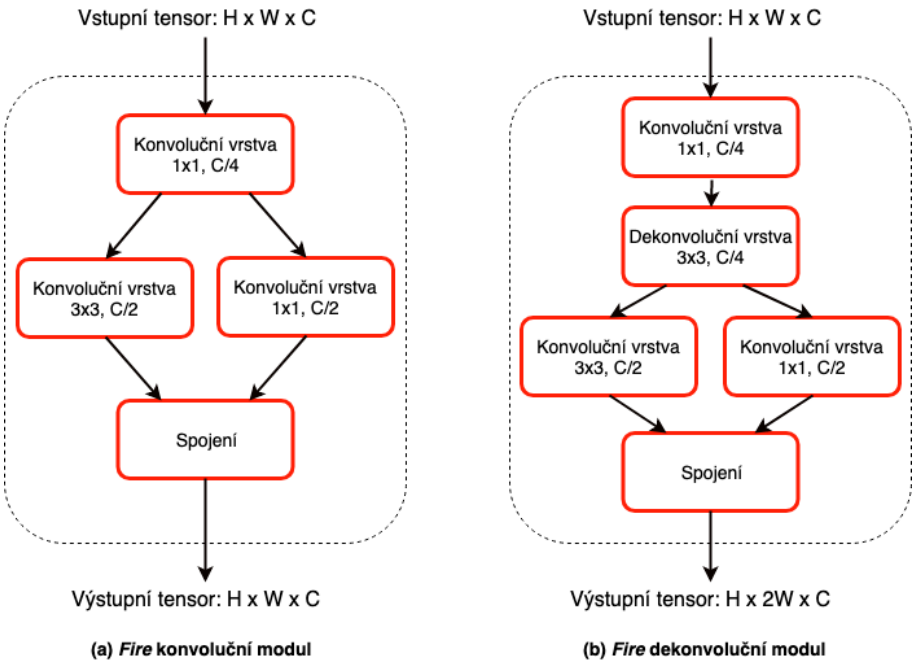
6.3.1 Fire moduly

Takzvané *fire moduly* jsou základním stavebním kamenem sítě SqueezeNet, jejíž autoři deklarují, že je náhradou za konvoluční vrstvy s kernelem $[3 \times 3]$ [15]. V modelu byly vyměněny konvoluční a dekonvoluční vrstvy, z důvodů úspory parametrů a výpočetního výkonu, právě za zmíněné *fire konvoluční* a *fire dekonvoluční* moduly. Jejich architektura je zobrazena na obrázku (18).

Ve *fire modulu* je neprve vstupní tensor o velikosti $H \times W \times C$ poslán do konvoluční vrstvy s kernelem 1×1 ke zredukování velikosti C čtyřikrát. Dále je použita 3×3 konvoluce k fúzi prostorové informace. Spolu s paralelní 1×1 konvoluční vrstvou se celková velikost C nezmění. Uvažujme stejný vstup u konvoluční vrstvy 3×3 a *fire modulu*. Klasická konvoluční vrstva

potřebuje $9C^2$ parametrů a musí vykonat $9HWC^2$ výpočtů, zatímco *fire modul* potřebuje pouze $\frac{3}{2}C^2$ parametrů a celkem musí provést pouze $\frac{3}{2}C^2$ výpočetních operací [14].

K převzorkování pomocí *fire dekonvolučního modulu* je mezi první konvoluční 1×1 vrstvu a dvě paralelní vrstvy vložena dekonvoluční. K zvýšení šířky W na dvojnásobek pomocí standardní dekonvoluční vrstvy je potřeba $4C^2$ parametrů a celkem je nutno vykonat $4HWC^2$ operací, kdežto *fire dokonvolučnímu modulu* stačí pouze $\frac{7}{4}C^2$ parametrů a $\frac{7}{4}HWC^2$ výpočtů [14].



Obrázek 18: Fire moduly (definovány v SqueezeNet [15])

6.4 Experimenty

V oblasti předzpracování dat je plně dostačující běžně dostupný, sériově vyráběný, hardware, jelikož operace, které nad daty v této fázi provádíme, nejsou příliš složité. Proto byl pro tyto účely použit laptop MacBook Pro 13,3", model 2015 s procesorem Intel i5-5257U, který je osazen integrovaným grafickým čipem Intel Iris 6100. Počítač má k dispozici 8 GB paměti RAM a pevný disk o velikosti 256 GB.

Abychom ale mohli otestovat funkčnost a správnost segmentace, musíme nejprve síť natrénovat. Již dříve jsme zmínili, že trénování konvolučních neuronových sítí je výpočetně a paměťově velice náročné. Na běžném počítači s integrovanou grafickou kartou by učení neuronové sítě bylo zdlouhavé, a proto usuzujeme, že učení na integrované grafické kártě je možné, ale vysoce nedoporučené. K dispozici ovšem máme školní server osazený operační pamětí o velikosti 64 GB, dvěma procesory Intel Xeon E5-2640 v2, které dohromady disponují šestnácti fyzickými jádry, kde každé má celkem dvě vlákna. Dále je server vybaven základní deskou X9DRG-QF společnosti Supermicro. To největší urychlení výpočtů zajišťuje masivní paralelizace pomocí dvou

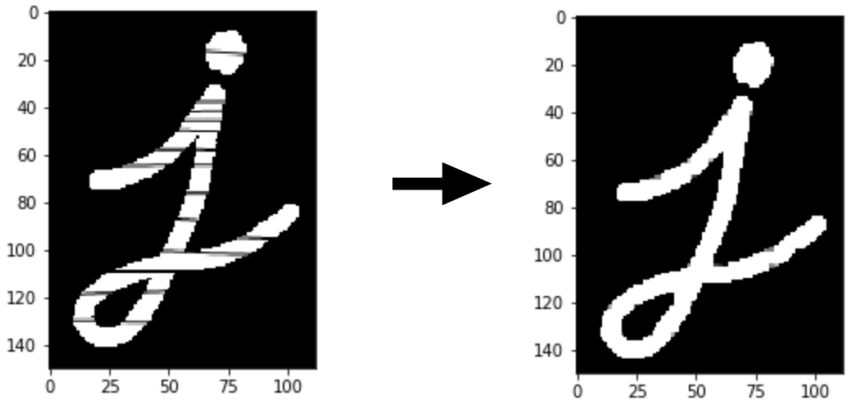
grafických karet kalifornské firmy NVIDIA GeForce RTX 2080 s podporou technologie CUDA. Každá z těchto grafických karet disponuje pamětí o velikosti 8 GB.

6.4.1 Předzpracování dat

Za jeden zpracovaný snímek považujeme proces načtení binárního souboru mračna, souboru se značkami, vyznačení bodů objektů v mračnu, selekce pouze potřebných bodů mračna, projekce bodů mračna na 2D mřížku a uložení výsledného tensoru do souboru. S použitím knihovny *NumPy* po zpracování všech snímků bez použití paralelizace jsme zjistili, že průměrná doba zpracování jednoho snímku je 26,717 ms. Pomineme-li operace načítaní a ukládání dat, jeden snímek je zpracován průměrně za 17,29 ms. Vzhledem ke specifikaci zařízení Velodyne HDL-64E je maximální frekvence 15 snímků/s, a to znamená, že zařízení vyprodukuje jeden snímek za 66 ms. Segmentace, včetně předzpracování jednoho snímku, by tedy neměla trvat déle, než zmíněných 66 ms, pokud je určena ke zpracování v reálném čase.

Výpočty také výrazně urychluje selekce bodů z předního výhledu vozidla. Průměrné mračno obsahuje přibližně 116 732 bodů, selekcí použitelných bodů dokážeme zredukovat velikost mračna průměrně na 30 168 bodů. Celkově jde o zmenšení větší než 74 %. Mračno je možné zredukovat také o body země, ale shlukovací algoritmy, které byly použity v avizovaných pracích v předchozí kapitole, nejsou dostatečně rychlé.

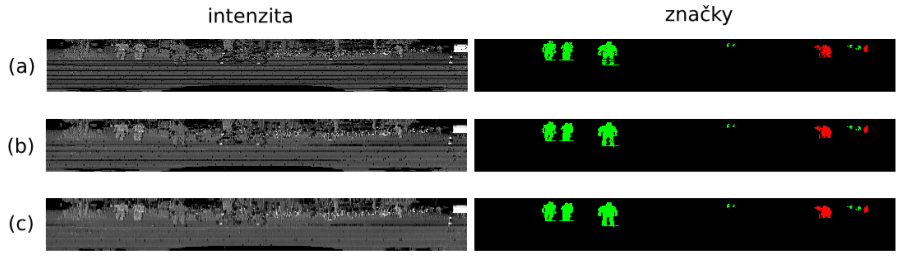
Pomocí naší metody předzpracování dat vznikají ve výsledných obrazech (16) vodorovné čáry, které považujeme za „šum“, vzniklý špatným angulárním rozlišením zařízení. Experimentálně jsme se zabývali odstraněním těchto čar pomocí morfologických transformací. Jedná se o funkce, kde prvním vstupním parametrem je 2D obraz, který chceme upravit. Druhým parametrem je strukturní prvek nebo kernel, na jejichž základě se upravuje obraz. Pro naše účely vyplnění „prázdných“ čar v obraze se nejlépe hodí operátor *uzavření*.



Obrázek 19: Příklad morfologického uzavření s kernelem [3 × 3] a s rozměry obrazu 112 × 150, kde vlevo je obraz původní a vpravo po třech iteracích uzavření.

V praxi je zobrazeno použití morfologické operace uzavření na obrázku (19). Byla použita implementace z knihovny *OpenCV*, která slouží k manipulaci s obrazem. Našim cílem je tedy vyhladit povrch obrazu a vyplnit vodorovné čáry. Je ale nutné, abychom zachovali tvary objektů, jelikož konvoluční neuronová síť často rozpoznává objekty na základě společných a podobných tvarů.

Aplikace na naše vstupní obrazy o velikosti 64×512 je vizualizována na obrázku (20). Z něj je patrné, že čáry při dvou iteracích částečně vymizely, ale došlo k mírné deformaci objektů. U tří iterací je to ještě patrnější a proto usuzujeme, že použití matematické morfologie zde není vhodné.

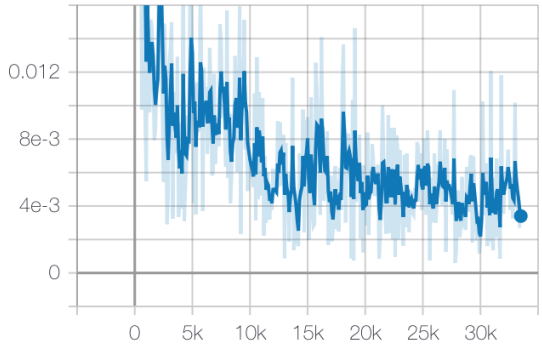


Obrázek 20: Příklad morfologického uzavření s kernelem $[1 \times 1]$ a s rozměry obrazů 64×512, kde řádek (a) obsahuje původní obrazy, řádek (b) jednu iteraci morfologického uzavření a poslední řádek (c) je aplikování mofologického uzavření dvakrát.

6.4.2 Učení neuronové sítě

Náš model sítě byl naučen na výše zmiňovaném školním serveru s pomocí knihovny TensorFlow ve verzi 1.13. Celkově byl model učen ve 32 000 krocích s velikostí vzorku 4 ve 32 epochách. Celkem jsme měli k dispozici 7481 vstupních snímků $64 \times 512 \times 5$ a ke každému také korespondující snímek se značkami. Koeficient učení byl nastaven na 0,001.

Vývoj učení modelu jsme sledovali pomocí nástroje TensorBoard. Konkrétně bylo sledováno: vývoj ztrátové funkce, přesnost segmentace pomocí metriky *IoU*, výsledné masky s predikovanými objekty a pravděpodobnostní mapy, ty poté potřebují aktivaci pomocí funkce softmax. Vývoj ztrátové funkce je zobrazen na obrázku (21). Příklady výsledné segmentace jsou zobrazeny na obrázku (22).



Obrázek 21: Vývoj ztrátové funkce. Vodorovná *osa x* označuje počet trénovacích kroků a svislá *osa y* označuje hodnotu ztrátové funkce. U funkcí tohoto typu platí, že menší hodnota je lepší.

6.4.3 Dosažené výsledky

K evaluaci našich výsledků byla použity metriky *IoU*, *precision* a *recall*, jejíž definice byly zmíněny již výše v textu. Po natrénování sítě na vzorku 5672 snímků, po celkem 32 000 učících krocích se hodnota ztrátové funkce zastavila na hodnotě 0,00340. Pro ověření správnosti segmentace se musí proces spustiti nad daty, které neuronová síť při učení nikdy neviděla, aby byly výsledky objektivnější. Po evaluaci všech testovacích dat byla hodnota ztrátové funkce 0,00710.

Průměrná hodnota *IoU* všech prvků testovací množiny po naučení sítě byla 31,87 %, *precision* dosahovalo průměrných hodnot 42,26 % a průměrná naměřená hodnota metriky *recall* byla 39,26 %. Dosažené výsledky pro jednotlivé druhy objektů jsou zobrazeny v následující tabulce, kde můžeme vidět, že segmentace automobilů je na poměrně dobré úrovni. U chodců a cyklistů jsou výsledky výrazně horší. Z veliké části je to zapříčiněno tím, že v celé množině dat jsou chodci zastoupeni pouze 222krát, cyklisté 1627krát a automobily zabírají největší část. Celkem se v LiDARových snímcích nacházejí 28742krát. Dalším problémem u cyklistů je tvar kol, které se špatně segmentují, protože jejich tvar je ve snímku často nespojitý a síť je může zaměňovat s chodci.

Metrika	**Automobil**	**Člověk**	**Cyklista**
IoU [%]	60,05	24	11,57
Precision [%]	71,7	35,1	20
Recall [%]	73,5	27	17,27

Tabulka 6: Dosažené průměrné výsledky segmentace na testovací množině dat (IoU, Precision, Recall)

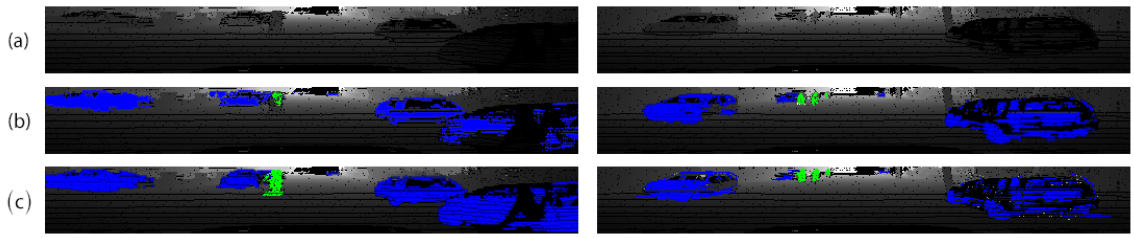
Základní metodiky určení kvality našeho klasifikátoru již byly stručně představeny dříve v této práci. Pro určení kvality jsme se rozhodli použít všech tří způsobů - *IoU*, *precision* a

recall. Výpočet metrik pro naše účely je dán následujícími vztahy

$$\begin{aligned} IoU &= \frac{TP}{TP + FN + FP}, \\ Pr &= \frac{TP}{TP + FP}, \\ Re &= \frac{TP}{TP + FN}, \end{aligned} \tag{26}$$

kde TP (pravdivě pozitivní) je počet správně určených bodů objektu, FP (falešně pozitivní) jsou body nesprávně označeny jako náležící objektu a FN (falešně negativní) je počet bodů, které byly chybně označeny, že nenáleží objektu. Výpočty byly aplikovány celkem na 1878 snímků testovací množiny.

Mezi další důležitý sledovaný aspekt je rychlost celého procesu. Již dříve jsme zmiňovali, že nechceme, aby zpracování jednoho snímku trvalo déle, než do doby, kdy je k dispozici nový snímek, což je v našem případě 66 ms. Při ověřování správnosti segmentace jsme také sledovali, jak rychle dokáže naučený model sítě vyprodukovat výsledek. Průměrná hodnota rychlosti zpracování jednoho snímku je 8,06 ms. Po přičtení doby předzpracování obrazu se dostáváme na hodnotu 25,35 ms, což je skvělý výsledek pro použití v asistenčních systémech a autonomně řízených automobilech.



Obrázek 22: Příklad výsledné segmentace, kde **(a)** je vstupní intenzita, **(b)** je predikovaný výsledek a **(c)** je předpokládaný výsledek.

7 Závěr

Výsledkem bakalářské práce je funkční klasifikátor automobilů, cyklistů a chodců z LiDARových dat. Celá práce a její experimenty byly realizovány v programovacím jazyce Python s pomocí knihovny pro strojové učení *TensorFlow*, kterou hodnotíme jako velmi šikovný nástroj pro rychlé sestavování modelů strojového učení.

Pro splnění práce v plném rozsahu bylo nutno nastudovat teorii nejen okolo neuronových sítí, ale i v oblasti předzpracování 3D dat a jejich projekci. Veškeré studium věnované této práci se nám jeví jako velice přínosné, jelikož obor autonomně řízených aut a umělé inteligence má dnes obrovský potenciál. Obzvláště pak s použitím LiDARových technologií, jejichž oblast není prozkoumána do takové míry, jako například zpracování obrazu.

Výsledná aplikace je rozdělena do tří částí - předzpracování dat, rozdělení dat a samotná neuronová síť. Skripty jsou napsány univerzálně tak, že lze například jednotlivé parametry sítě jednoduše měnit. Model natrénované sítě je vhodný pro použití v *real-time* klasifikaci objektů z LiDARových dat, jelikož procesy sémantické segmentace a předzpracování dat jsou dostatečně rychlé. Skript předzpracování dat je připraven pouze pro data z KITTI [2], nicméně nebude vyžadovat veliké úsilí, aby byl přepsán do podoby, ve které bude přijímat jiný formát dat. Také je možné použít LiDARová data i jiných zařízení, než je Velodyne *HDL-64E*, například *VLP-16*. Každopádně je nutné počítat se snížením účinnosti, jelikož vertikální rozlišení je oproti prvně zmiňovanému typu čtyřikrát menší. To má za následek, že mračno nebude tak husté a objekty budou obsahovat méně detailů, na základě kterých se klasifikátor může rozhodovat hůře.

Z uvedení několika algoritmů určených pro segmentaci a klasifikaci v mračnech vyplývá, že předzpracování dat, stejně jako model sítě, nabízí obrovský prostor pro experimenty. Můžeme vyrobit více dat mírným upravením těch původních, odebrat body země, nebo také přidat další vrstvy do modelu. Každopádně po srovnání mnoha dostupných metod je zřejmé, že dnes se sémantická segmentace v mračnech bodů se ubírá směrem konvolučních neuronových sítí.

Seznam zdrojů

1. STEVE JURVETSON. *Lexus RX450h retrofitted by Google for its driverless car fleet. At the left side is parked a Tesla Model S electric car.* 2012. Dostupné také z: `https://commons.wikimedia.org/wiki/File:Driving_Google_Self-Driving_Car.jpg`. [Online; navštíveno 4. dubna, 2019].
2. GEIGER, Andreas; LENZ, Philip; STILLER, Christoph; URTASUN, Raquel. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*. 2013.
3. ANN NEAL. *LiDAR vs. RADAR*. 2018. Dostupné také z: `https://www.sensorsmag.com/components/lidar-vs-radar`. [Online; navštíveno 3. dubna, 2019].
4. VELODYNE LIDAR, INC. *Velodyne LiDAR HDL-64e*. 2019. Dostupné také z: `https://velodynelidar.com/hdl-64e.html`. [Online; navštíveno 3. dubna, 2019].
5. TYLER LEE. *A Self-Driving Car's Laser Reportedly Destroyed A Man's Camera*. 2019. Dostupné také z: `https://www.ubergizmo.com/2019/01/self-driving-car-laser-destroy-camera/`. [Online; navštíveno 20. dubna, 2019].
6. THOMAS PAUL. *LiDAR Basics: The Coordinate System*. 2018. Dostupné také z: `https://hackernoon.com/lidar-basics-the-coordinate-system-a26529615df9`. [Online; navštíveno 4. dubna, 2019].
7. POINTCLOUDS.ORG. *The Velodyne High Definition LiDAR (HDL) Grabber*. 2017. Dostupné také z: `http://pointclouds.org/documentation/tutorials/hdl_grabber.php`. [Online; navštíveno 3. dubna, 2019].
8. KUPKA, Jiří. *Konvoluční neuronové sítě v analýze obrazu*. Ostrava, 2016. Diplomová práce. VŠB - Technická univerzita Ostrava.
9. KAČER, Petr. *Konvoluční neuronové sítě a jejich implementace*. Brno, 2013. Bakalářská práce. Vysoké učení technické v Brně.
10. WIKIPEDIE, OTEVŘENÁ ENCYKLOPEDIE. *Princip diskrétní dvourozměrné konvoluce*. 2006. Dostupné také z: `https://cs.wikipedia.org/wiki/Konvoluce#/media/File:Konvoluce_2rozm_diskretni.jpg`. [Online; navštíveno 5. dubna, 2019].
11. SCHMID, Martin. *Konvoluční neuronové sítě a jejich implementace*. Praha, 2011. Bakalářská práce. Univerzita Karlova.
12. SHYAMAL PATEL AND JOHANNA PINGEL. *Convolutional Neural Network*. 2017. Dostupné také z: `https://www.sensorsmag.com/components/lidar-vs-radar`. [Online; navštíveno 2. dubna, 2019].
13. ASVADI, Alireza; GARROTE, Luís; PREMEBIDA, Cristiano; PEIXOTO, Paulo; NUNES, Urbano. DepthCN: Vehicle detection using 3D-LIDAR and ConvNet. In: 2017. Dostupné z DOI: `10.1109/ITSC.2017.8317880`.

14. WU, Bichen; WAN, Alvin; YUE, Xiangyu; KEUTZER, Kurt. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. *ICRA*. 2018.

15. IANDOLA, Forrest N.; MOSKEWICZ, Matthew W.; ASHRAF, Khalid; HAN, Song; DALLY, William J.; KEUTZER, Kurt. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*. 2016, roč. abs/1602.07360. Dostupné z arXiv: 1602.07360.

16. WANG, Yuan; SHI, Tianyue; YUN, Peng; TAI, Lei; LIU, Ming. PointSeg: Real-Time Semantic Segmentation Based on 3D LiDAR Point Cloud. *CoRR*. 2018, roč. abs/1807.06288. Dostupné z arXiv: 1807.06288.

17. ZHOU, Yin; TUZEL, Oncel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. *CoRR*. 2017, roč. abs/1711.06396. Dostupné z arXiv: 1711.06396.

18. GOOGLE LLCL. *Tensorflow.org*. 2019. Dostupné také z: https://www.tensorflow.org/. [Online; navštíveno 16. dubna, 2019].

8 Obsah přiloženého archivu

- **thesis.pdf** - Elektronická verze práce.
- **lidar-object-recognition/**Složka se zdrojovými kódy a modelem sítě.
  - **README.md** - Návod ke spuštění skriptů ve složce **src/**.
  - **src/** - Složka obsahující zdrojové kódy a model sítě.
    - **model/** - Složka, kde se nachází naučený model sítě knihovnou TensorFlow.
    - **experimental/** - Složka se zdrojovými kódy pro zkoumání experimentálních částí programu.
    - **lib/** - Složka s knihovnami třetích stran.
    - **train.py** - Skript pro trénování a evaluaci dat.
    - **predict.py** - Skript určený pro predikci jednoho konkrétního snímku z libovolné množiny dat.
    - **preprocess.py** - Skript sloužící k předzpracování dat do požadovaného formátu.
    - **split_data.py** - Skript, který rozděluje data dle požadovaného poměru.
  - **example/** - Složka, ve které lze nalézt vícero vizualizací vstupních vrstev CNN a příklady výsledné segmentace.
- **requirements.txt** - Soubor obsahující názvy a verze knihoven, které je potřeba nainstalovat pro správnou funkčnost.